14p

Massachusetts Institute of Technology
Cambridge, Massachusetts
Project MAC

Artificial Intelligence Project
Memo 68

Memorandum MAC-M-158
May 29, 1964

Syntax of the New Language

by Michael Levin

This is a definition of the syntax of the *** language. It consists of modifications and extensions of the Revised Report on the Algorithmic Language ALGOL 60 which is printed in the Communications of the ACM, January 1963. The paragraph numbering of that report is used in this paper. The corrections and additions are made partially in Backus normal form, and partially in English, and the choice has been made on the basis of convenience. For example, the use of the weak separator ⟨space⟩ is described readily in a few sentences, whereas the modification to incorporate this into the syntax as described in Backus normal form would have been extensive.

This paper is not intended as a definitive description of the *** language. It is intended as a definition of a well formed program. Thus any *** program must belong to this generative grammar, but many instances of text belonging to this generative grammar will be semantically meaningless.

It is hoped that this grammar is free of syntactic ambiguity. If you find that this is not so, please call it to my attention. Correction sheets will be maintained for this paper.

The only intended omission is the syntactic definition of the ⟨comit rule⟩ , which will be given later. Input/output and other system procedures do not require syntactic extensions to the language.

Certain new constructions have brief semantic discussions, however, no effort has been made to systematically update the semantics paragraphs of the Algol 60 report. More complete semantics of the language will be supplied in later memos.

2.2.1 add

⟨octal digit⟩ ::= 0|1|2|3|4|5|6|7

2.3.1 Reserved Words

own, global, local, multiple

procedure

true, false

value, loc, functional, formal

array, matrix, individual

all, are

begin, end

Boolean, logical, real, integer, symbol

comment, meta

for, step, until, while, in, do

if, then, else

go, to

switch, label

nil, lambda, cond, quote, return

Reserved words are not distinguished by being written in bold face or other unusual type. This list does not include system procedures.

2.3.2 Special Characters

< ≤ > ≥ = ≠

∧ ∨ ~ ⊃ ≡

+ - × * / ÷ \ ↑

←

→ $

λ

' " ↑

( ) [ ] { }

. , : ;

2.3.3 The Space

⟨space⟩ ::= | ⟨space⟩

2.3.4 The Question Mark and Meta

The special character ? has the same significance as the reserved word comment. The reserved word meta is used to define extensions to the grammar within a program.

2.4.1 add

An identifier that is a reserved word may not be used as a label, variable, procedure identifier, or a formal parameter.

2.5.1 replace the entire paragraph

⟨unsigned decimal⟩ ::= ⟨digit⟩ | ⟨unsigned decimal⟩ ⟨digit⟩

⟨simple decimal⟩ ::= ⟨unsigned decimal⟩ | + ⟨unsigned decimal⟩ | - ⟨unsigned decimal⟩

⟨decimal integer⟩ ::= ⟨simple decimal⟩ | ⟨simple decimal⟩ I | ⟨simple decimal⟩ I ⟨unsigned decimal⟩

⟨unsigned octal⟩ ::= ⟨octal digit⟩ Q | ⟨octal digit⟩ Q ⟨unsigned decimal⟩ | ⟨octal digit⟩ ⟨unsigned octal⟩

⟨octal integer⟩ ::= ⟨unsigned octal⟩ | + ⟨unsigned octal⟩ | - ⟨unsigned octal⟩

☆ ⟨integer⟩ ::= ⟨octal integer⟩ | ⟨decimal integer⟩

⟨unsigned real⟩ ::= ⟨unsigned decimal⟩ . | . ⟨unsigned decimal⟩ | ⟨unsigned decimal⟩ . ⟨unsigned decimal⟩

⟨simple real⟩ ::= ⟨unsigned real⟩ | + ⟨unsigned real⟩ | - ⟨unsigned real⟩

⟨exponent part⟩ ::= E | E ⟨decimal integer⟩

☆ ⟨real⟩ ::= ⟨simple real⟩ | ⟨simple real⟩ ⟨exponent part⟩ | ⟨simple decimal⟩ ⟨exponent part⟩

2.5.2 replace the entire paragraph with

Correct Examples

0	-200.084	-.083E-02
177	+07.43E8	-1I7
.5384	9.34E+10	1.E-4
+0.7300	2.E-4	+1I5
43Q	6Q9	77I3
9E	2.4E0	0E4
3E6	4E+2I7	4E-3

Incorrect Examples

E4	39Q	3I4I5I6
-E+3	.E3	

2.5.3 replace with

E and I indicate exponentiation to the base 10.

Q indicates exponentiation to the base 8.

No octal integer contains an 8 or 9 to the left of the Q.

The exponent is always a decimal integer.

The exponent of an integer is always unsigned.

A real number always contains a decimal point, or the letter E.

A real number always contains at least one digit not part of the exponent.

2.6 through 2.8 delete and replace with

2.6 Strings

A string is any sequence of characters. Any character in the available hardware may be used including even those used as special signals such as carriage return.

A quoted string is a string with two quote marks (") immediately preceding and following it. Any occurrence of the quote mark within a string must be replaced by ?" when the string is quoted. Similarly, the question mark must be replaced with ?? when the string is quoted. Characters that can appear in output only may be represented by the question mark followed by some character which is the name of that character.

2.6.1 Examples

unquoted	quoted
abc	"abc"
xy ,(z	"xy ,(z"
3.*"y	"3.*?"y"
??"" "	"?????"?" ?""

Strings are a basic constituent of symbolic expressions. Unlike identifiers, they do not have property lists, and are not uniquely represented. Each copy of the string carries its own character representation.

When a string appears within a program, it is always quoted. When strings are read in from data using formatted input, quoting is not needed.

2.7 Symbolic Expressions

⟨atom⟩ ::= ⟨identifier⟩ | ⟨quoted string⟩ | ⟨logical value⟩ |
⟨integer⟩ | ⟨real⟩ | ⟨array⟩

⟨spacer⟩ ::= | ⟨spacer⟩ | , | , ⟨spacer⟩

⟨S-list⟩ ::= ⟨S-expression⟩ | ⟨S-expression⟩ ⟨spacer⟩ ⟨S-list⟩

⟨S-expression⟩ ::= ⟨atom⟩ | (⟨S-expression⟩ ⟨spacer⟩ . ⟨spacer⟩
⟨S-expression⟩) | (⟨S-list⟩) | ( )

⟨quoted S-expression⟩ ::= ' ⟨S-expression⟩ | ⟨quoted string⟩

A single mark at the beginning of an S-expression is sufficient to quote it since an S-expression is self-limiting by its own parentheses. In the case of an S-expression which is a single string, it is redundant to use both the single and the double quote, although not prohibited.

2.7.1 Examples

unquoted	quoted
a	'a
"a"	'"a" or "a"
(3. 4.0 7 true)	'(3. 4.0 7 true)
(2 A "$")	'(2 A "$")
(4 . 3)	'(4 . 3)
(4.3,6.2)	'(4.3,6.2)

S-expressions must be quoted when they occur within a program. They do not need to be quoted when read as data, even when the input is unformatted. However, strings that occur within non-atomic S-expressions always require the string quote.

2.8 The Array

It is possible to express constant arrays within a program.

Example:

$\begin{pmatrix} 3 & 2 \\ 4 & -7 \end{pmatrix}$ is written as [[3,2],[4,-7]]

Each sublevel of an array is represented as a list with square brackets as delimiters and commas as separators. There are as many sublevels as there are dimensions to the array. The innermost level corresponds to the first dimension in the array subscripting. Each list must be of the proper length for its dimension.

The elements of an array may be any type of unquoted symbolic expression, except arrays (which must be quoted using the single quote mark). Included as possible array elements are numbers, logical value, and quoted strings.

This notation is provided for the sake of completeness. Constant arrays will usually be read from data using formatted input.

All arrays written using this notation have lower subscript bounds of 1.

2.9 Data

⟨datum⟩ ::= ⟨integer⟩ |⟨real⟩ |⟨logical value⟩ |⟨array⟩ |⟨quoted S-expression⟩

2.10 The Space

2.10.1 The ⟨space⟩ must never be used within an identifier, within a reserved word, or within a number. If it is used within a quoted string it will be taken literally.

2.10.2 It is permissible to use the ⟨space⟩ at any place within a program except those specified in 2.10.1 without changing the meaning of the program.

2.10.3 The ⟨space⟩ must be used whenever its omission would result in the concatenation of any two items each of which is a number, an identifier, or a reserved word. The ⟨space⟩ is not required if any special character comes between the two items.

2.10.4 Examples

Correct:

```
loop: if a=3 then go to loop else go to end;
```

Incorrect:

```
lo op: ifa =3then goto loop elsegotoe nd;
```

Correct:

```
loop            :if a=  3   then   go
                to loop    else go to end
                                        ;
```

3. Expressions

3.1 change ⟨subscript expression⟩ := ⟨expression⟩

3.2 replace with

Procedure Designators

⟨procedure identifier⟩ ::= ⟨identifier⟩

⟨identifier list⟩ ::= ⟨identifier⟩ | ⟨identifier⟩ , ⟨identifier list⟩

⟨formal parameter list⟩ ::= ; | ⟨identifier list⟩;

⟨declaration list⟩ ::= ⟨empty⟩ | ⟨declaration⟩ ; | ⟨declaration⟩ ;
⟨declaration list⟩

⟨procedure body⟩ ::= ⟨statement⟩ | ⟨expression⟩

⟨procedure designator⟩ ::= ⟨procedure identifier⟩ | λ ⟨formal parameter list⟩ ⟨declaration list⟩ ⟨procedure body⟩

3.3 Expressions

⟨infix operator⟩ ::= +|-|×|*|/|÷|\|↑|<|≤|>|≥|=|≠|∧|∨|⊃|≡

⟨prefix operator⟩ ::= -|~

⟨argument list⟩ ::= ⟨expression⟩ | ⟨expression⟩ , ⟨argument list⟩

⟨simple expression⟩ ::= ⟨datum⟩ | ⟨variable⟩ | ⟨simple expression⟩
⟨infix operator⟩ ⟨simple expression⟩ | ⟨prefix operator⟩
⟨simple expression⟩ | ⟨procedure designator⟩ (⟨argument list⟩) |
⟨procedure designator⟩ ( ) | (⟨expression⟩)

⟨expression⟩ ::= ⟨simple expression⟩ | if ⟨expression⟩ then ⟨simple expression⟩ else ⟨expression⟩

Every procedure including those designated by infixes and prefixes

must have a type for its arguments and for its value. Appropriate transfer functions may automatically be invoked in certain cases. It is expected that an expression following the reserved word if will be Boolean valued, but this is not part of the syntax of ***.

When the grouping of an expression using prefix and infix operators is not specified by the use of parentheses, all operators will have their scope determined by a precedence table, with those of lowest precedence having the widest scope. New operators may be added to the tables when in the meta mode. The precedence level of an infix operator must be specified. The initial precedence table is:

≡	< ≤ > ≥ = ≠
⊃	+ −
∨	× * / ÷ \
∧	↑
~	

The symbol * is for matrix multiplication, and the symbol is for integer remainder.

3.5 change

⟨label⟩ ::= ⟨identifier⟩

The rest of 3.5 is as in the revised Algol report, but note that a designational expression is no longer an expression.

The integer label has been deleted. The switch is retained as the only way of computing the location of a jump at run time.

4.1.1 add

The braces { , and } are equivalent to the reserved words begin and end respectively.

4 2.1 change

⟨left part⟩ ::= ⟨variable⟩ | ⟨procedure identifier⟩

If a procedure designator is assigned to a procedure identifier, the result of this is to change the definition of the procedure.

Assignment of a value to a procedure identifier within that procedure is never used to return a value for the procedure. The proper way to do this is with return as in the LISP procedure.

An assignment statement may be considered as an expression having the value assigned to the variable of the left half of the expression. The left arrow has a precedence level that is very high on the left and very low on the right.

Example:

a ← 3 + b ← 4

is equivalent to

a ← (3 + (b ← 4))

4 6.1 replace entire paragraph with

The for statement is being enlarged, and the extent of this enlargement has not been decided at present. It will contain the Algol 60 for statement, and may have the following additional properties:

1. The inclusion of reset in addition to step. This reassigns the variable instead of adding to it.

2. A notation for permeating a symbolic expression or travelling the length of a list with the control variable set to each subexpression in turn.

3. Several control variables being simultaneously updated.

4. The ability to indicate whether testing occurs at the beginning or or the end of the loop.

4.7.1 replace entire paragraph with

⟨procedure statement⟩ ::= ⟨procedure designator⟩ | ⟨procedure designator⟩( ) | ⟨procedure designator⟩(⟨argument list⟩)

5. replace entire paragraph with

⟨declaration⟩ ::= ⟨variable declaration⟩ | ⟨switch declaration⟩ | ⟨default declaration⟩ | ⟨procedure declaration⟩

5.1 Variable Declaration

⟨transmission mode⟩ ::= value | loc | formal | functional

⟨type⟩ ::= real | integer | Boolean | logical | symbol

⟨storage mode⟩ ::= local | global | own

⟨size⟩ ::= array | matrix | individual

⟨specifier⟩ ::= ⟨transmission mode⟩ | ⟨type⟩ | ⟨storage mode⟩ | ⟨size⟩ | ⟨function⟩ ⟨procedure⟩ | ⟨specifier⟩ ⟨specifier⟩

⟨lower bound⟩ ::= ⟨expression⟩

⟨upper bound⟩ ::= ⟨expression⟩

⟨bound⟩ ::= ⟨upper bound⟩ | ⟨lower bound⟩ : ⟨upper bound⟩

⟨bound list⟩ ::= ⟨bound⟩ | ⟨bound⟩ , ⟨bound list⟩

⟨item⟩ ::= ⟨identifier⟩ | ⟨identifier⟩ ← ⟨expression⟩ | ⟨identifier⟩ [⟨bound list⟩]

⟨variable declaration⟩ ::= ⟨specifier⟩ ⟨item⟩ | ⟨variable declaration⟩ , ⟨item⟩

These declarations specify the type of a variable, whether it is an array or matrix, its dimensions if applicable, its storage mode, its mode of transmission if it is a parameter, and whether or not it is multiple if it is a parameter. Any of this information can be grouped within one

specifier, but most of it is usually redundant and can be omitted. The assignment within a declaration is an initialization of the value. If not given, initial values will be false,0,0.0, or nil depending on the type.

5.2 Switch Declarations

See 5.3 of the revised Algol report.

5.3 Default declarations

⟨default declaration⟩ ::= all ⟨specifier⟩ are ⟨specifier⟩

A default declaration govers the properties of variables when full information is not given. It asserts that anything having all of the properties of the left half will have all of the properties of the right half unless otherwise specified.

If no default declarations are present then:

1. All procedures of unspecified type are symbol.

2. All variables of unspecified type are the type of their procedure.

3. All variables are individual unless otherwise specified.

4. All individual parameters are transmitted by value unless otherwise specified.

5. All matrix and array variables are transmitted by location unless otherwise specified.

6. All variables are local unless otherwise specified.

5.4 Procedure Declarations

⟨procedure type⟩ ::= ⟨type⟩ | ⟨empty⟩ |⟨type⟩⟨matrix⟩ |⟨type⟩ ⟨array⟩|
⟨matrix⟩|⟨array⟩

procedure declaration ::= proceudre type procedure procedure

⟨procedure declaration⟩ ::= ⟨procedure type⟩ procedure |
⟨procedure type⟩ procedure ⟨procedure identifier⟩ ⟨procedure designator⟩ |
⟨procedure type⟩ procedure ⟨procedure identifier⟩
⟨formal parameter list⟩ ⟨declaration list⟩ ⟨statement⟩ |
⟨procedure type⟩ procedure ⟨procedure identifier⟩
⟨formal parameter list⟩ = ⟨declaration list⟩ ⟨expression⟩

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # AIM-68

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 14 (18-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☒ Single-sided or
☐ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other: MIMEOGRAPH

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): _____

Photographs/Tonal Material (by page number): _____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP:	(1-14)
UNN'ED TITLE PAGE,	2-14
SCANCONTROL, TRGT'S	(15-18) (3)

Scanning Agent Signoff:

Date Received: 11/30/95 Date Scanned: 12/6/95 Date Returned: 12/7/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**